Escuela de Ingeniería en Computación
Ingeniería en Computación
IC6600 - Principios de Sistemas Operativos

Informe Proyecto 1

Reyner Marxell Arias Muñoz
reyner.arias@estudiantec.cr
2019156290

David Benavides Naranjo
davidbn42@estudiantec.cr
2019018297

Kenneth Ibarra Vargas
kiv@estudiantec.cr
2019035508

San José, Costa Rica
23 de Setiembre del 2022

Índice general

Capítulo 1

Introducción

Este documento corresponde al informe del Proyecto 1 de curso de Principios de Sistemas Operativos. Algunos de los conceptos presentes a lo largo de esta asignación son: hilos, regiones críticas, condiciones de carrera y sincronización. Primero, los hilos corresponden a una ruta de ejecución dentro un proceso, este último puede contener varios hilos y estos comparten los registros y la dirección de memoria del proceso al que pertenecen (GeeksforGeeks, 2021). Después, tenemos las regiones críticas que son segmentos de código que varios hilos tratan de acceder al mismo tiempo. Estos usualmente suelen contener variables compartidas que deben ser sincronizadas si se desean evitar resultados impredecibles. Además, estas regiones se dice que deben ser fragmentos atómicos de código, es decir, no se pueden dividir en partes más pequeñas (GeeksforGeeks, 2022).

Luego, debido al mal control de las regiones críticas ocurren las condiciones de carrera, problemática relacionada con el acceso simultáneo de dos hilos a un recurso. Especifícamente, pueden surgir debido a que un hilo sobreescribe el trabajo realizado por otro y este último no se da cuenta, leyendo posteriormente un resultado erróneo (Yu, 2022). Por último el documento utiliza el concepto de sincronización de hilos, el cual se refiere a controlar el acceso de múltiples hilos a recursos compartidos (Javatpoint, s.f.).

1.1. Github

Todo el código y archivos solicitados en la especificación de la tarea se encuentran en el siguiente git: https://github.com/RyutaKinzoku/RogueThread.

Capítulo 2

Desarrollo

2.1. Regiones críticas presentes en el juego y posibles condiciones de carrera

Dado que este proyecto no es trivialmente paralelizable, podemos identificar varias regiones críticas que son accedidas por los diferentes hilos del juego, los dos hilos de héroe y un hilo por cada monstruo, estas son:

- **entityMap** Este recurso corresponde a una matriz de enteros que se encarga de representar la posición de los monstruos en el mapa. Si el valor en cierta posición es 0, no hay un mostruo en esa celda, por otro lado los enemigos son representados con valores del 1 a n/2 para darles una identificación. Estas casillas son propensas a condiciones de carrera debido a que dos monstruos podrían intentar entrar en la misma sala en un momento dado, traslapando sus tareas y generando comportamientos inesperados en la aplicación. Eso es un fallo en la lógica del programa ya que la especificación dicta que solo puede existir un monstruo por cuarto.

- **heroHealth** Esta variable de tipo entero representa la vida actual del jugador y corresponde a una sección crítica en el programa debido a que existen dos formas de reducir la vida, el ataque de un enemigo y la caída en una trampa, acciones realizadas por distintos hilos de ejecución. La disminución de vida por ataque es realizada por algún monstruo y la caída en una trampa por el hilo del héroe. La principal condición de carrera ocurre cuando un hilo intenta reducir este valor y se descalendariza en una instrucción de ensamblador intermedia de la operación de decremento de C, después otro hilo procesa una reducción exitosamente e invade los registros que estaba utilizando el primer hilo, esto puede causar un valor inesperado en la vida del héroe.

2.2. Estructuras de sincronización

En la aplicación utilizamos principalmente una estructura de exclusión mutua (health_mutex) para la vida del héroe y una lista de exclusiones mutuas (cells_mutex) para cada celda del mapa. “La funcionalidad de los objetos de exclusión mutua es permitir este uso de un solo recurso mediante la creación de un entorno en el que el acceso al recurso se transfiere continuamente entre los diversos aspectos del programa”(Spiegato, s.f.).
Como se menciono anteriormente, existen una serie de condiciones de carrera relacionadas con las regiones críticas de esta aplicación, por lo que la exclusión mutua ayuda a mantener estos fragmentos de código como una sola pieza que se debe ejecutar de forma serial. Por otro lado, utilizamos varias estructuras de join al final del hilo principal que se encargan de sincronizar todos los hilos y finalizar el programa correctamente. En resumen, “Esta llamada hace que el hilo se ‘duerma’ hassta que el otro hilo termine”(Chuidiang, s.f.).

2.3. Makefile

El makefile compila correctamente el programa en un ambiente Linux.

Capítulo 3

Conclusiones

La espera pasiva implementada para la recepción de comandos (wait/sleep) nos ahorra mucha capacidad de procesamiento dado que este no pregunta indefinidamente por el acceso a un recurso, más bien espera hasta que el usuario pulsa una tecla. Nos pareció beneficioso hacerlo de esta manera, ya que a través de los cursos anteriores cuestiones similares no tenían importancia y en el mundo laboral si se trabaja con programación multihilo es fundamental no desperdiciar recursos del procesador, incluso más si la programación concurrente fue inventada precisamente para evitarlo.

Por otro lado, aprendimos que en problemas de este tipo es complicado identificar las regiones críticas presentes y determinamos que en aplicaciones reales es una parte fundamental del diseño e implementación de aplicaciones con comportamiento asícrono. Por eso mismo, los equipos de trabajo deben gastar una cantidad considerable de tiempo realizando tareas de este tipo, y así evitar fallas de Software como las que provocan consideraciones de diseño incorrectas.

Por último, las condiciones de carrera son aleatorias y en algunos casos pocos probables, pero esto no nos puede llevar a pensar que no son importantes. Si tenemos ese tipo de pensamiento irresponsable y estamos trabajando en Software que vela por la vida humana podemos tener consecuencias tremendamente graves. Evidentemente, las consecuencias de este proyecto estan relacionadas con cuestiones de aprendizaje, un juego no pone en riesgo la vida humana o la vida de algún ser vivo.

Referencias

Chuidiang. (s.f.). *Algunos detalles sobre los threads.* Descargado de https://www.chuidiang.org/clinux/procesos/mashilos.php#:~:text=El%20hilo%20que%20espera%2C%20debe,funci%C3%B3n%20pthread_join()%20sale%20inmediatamente.

GeeksforGeeks. (2021). *Thread in operating system.* Descargado de https://www.geeksforgeeks.org/thread-in-operating-system/

GeeksforGeeks. (2022). *Critical section in synchronization.* Descargado de https://www.geeksforgeeks.org/g-fact-70/

Javatpoint. (s.f.). *Synchronization in java.* Descargado de https://www.javatpoint.com/synchronization-in-java

Spiegato. (s.f.). *¿qué es un mutex?* Descargado de https://spiegato.com/es/que-es-un-mutex

Yu, H. (2022). *Race conditions and deadlocks.* Descargado de https://learn.microsoft.com/en-us/troubleshoot/developer/visualstudio/visual-basic/language-compilers/race-conditions-deadlocks